

**TierDeveloper<sup>™</sup>**  
**Programmer's Guide for .NET**

1.	INTRODUCTION-----	3
2.	GENERATED CODE ARCHITECTURE -----	3
2.1.	NAMESPACES AND CLASS DESCRIPTION -----	3
2.1.1.	<i>Business Namespace</i> -----	4
2.1.2.	<i>Data.Persistence Namespace</i> -----	5
2.1.3.	<i>Data.Hooks Namespace</i> -----	5
2.1.4.	<i>Integration Namespace</i> -----	6
2.1.5.	<i>Webservices</i> -----	6
2.1.6.	<i>.NET Remoting</i> -----	7
2.1.7.	<i>Client Applications</i> -----	7
2.2.	TDEVFRAMEWORK LIBRARY-----	7
3.	WORKING WITH TIERDEVELOPER COMPONENTS-----	7
3.1	USING GENERATED COMPONENTS-----	7
3.2	CLASS USAGE-----	9
3.2.1.	<i>Domain Object Class</i> -----	9
3.2.2.	<i>Factory Class</i> -----	9
3.2.3.	<i>Collection Class</i> -----	11
3.3.	STANDARD METHODS -----	11
3.3.1.	<i>Load</i> -----	12
3.3.2.	<i>Insert</i> -----	12
3.3.3.	<i>Update</i> -----	12
3.3.4.	<i>Delete</i> -----	12
3.3.5.	<i>Save</i> -----	13
3.3.6.	<i>Delete Collection</i> -----	13
3.4.	QUERIES -----	13
3.4.1.	<i>Count Query</i> -----	13
3.4.2.	<i>Distinct Query</i> -----	14
3.4.3.	<i>Partial Query</i> -----	14
3.4.4.	<i>Parameterized Query</i> -----	14
3.4.5.	<i>Dynamic Query</i> -----	14
3.4.6.	<i>Joins (Inner/Outer)</i> -----	15
3.5.	STORED PROCEDURES -----	16
3.4.1.	<i>Stored Procedure Returning Rowset</i> -----	17
3.4.2.	<i>Stored Procedure Returning Collection</i> -----	17
3.5.	OPERATIONS -----	17
3.6.	BULK OPERATIONS-----	19
3.6.1.	<i>Bulk Update Operation</i> -----	19
3.6.2.	<i>Bulk Delete Operation</i> -----	19
3.7.	RELATIONSHIPS -----	20
3.7.1.	<i>One-to-One / Many-to-One Relationship</i> -----	20
3.7.2.	<i>One-to-Many Relationship</i> -----	20
3.7.3.	<i>Many-to-Many Relationship</i> -----	21
3.8.	MANAGING CONNECTION STRING -----	22
3.8.1	<i>Dynamic Connection</i> -----	22
3.8.2	<i>Custom Connection</i> -----	22
3.9	SUPPORT FOR NULLABLE TYPES -----	23
3.10	DATASETS -----	24
3.10.1	<i>Updateable DataSets</i> -----	25
3.11	OBJECT BINDING -----	27
3.11.1	<i>Simple Binding</i> -----	27

3.11.2	<i>Complex Binding</i>	28
3.12	CODE CUSTOMIZATION	28
3.12.1	<i>Custom Hooks</i>	28
3.12.2	<i>Partial Classes</i>	29
3.12.3	<i>Template Customization</i>	29
3.13	LAZY LOADING	30
3.14	OBJECT INHERITANCE	31
3.14.1	<i>Vertical inheritance mapping</i>	31
3.14.2	<i>Filtered inheritance mapping</i>	33
3.15	TRANSACTIONS	33
3.15.1	<i>With Related Objects</i>	33
3.15.2	<i>With Unrelated Objects</i>	34
3.16	WEB SERVICES	36
3.17	.NET REMOTING	37
3.18	SERVICED COMPONENTS	37
3.18.1	<i>Dual Interface</i>	37
3.19	AJAX SUPPORT	38
3.20	MISCELLANEOUS	38
3.20.1	<i>Collection Sorting</i>	38
3.20.2	<i>Custom Formula Field</i>	39
3.20.3	<i>Import/ Export Support</i>	39
3.21	COMPONENT DEPLOYMENT	40
3.21.1	<i>Serviced Components</i>	40
3.21.2	<i>Non-Serviced Components</i>	40
3.22	REMOTING APPLICATION DEPLOYMENT	40
3.23	WEBSERVICES APPLICATION DEPLOYMENT	40

# 1. Introduction

This document is intended for software developers who would like to work with code generated by TierDeveloper. It includes discussion on the use of various features of the generated code, essential programming constructs and example code to use these features effectively. In order to fully benefit from this guide, you should be well familiar with the TierDeveloper Designer and different GUI elements used in the code generation process and utilize them to make the generation process a success. If not, we recommend you reading Online Help and Code Generation Overview, first.

## 2. Generated Code Architecture

In TierDeveloper latest version, the component's framework is specifically enhanced to cater performance and scalability factors. TierDeveloper generates code based upon industry-use standard design patterns for writing middle tier objects. Let us first look into the generated code design before delving deep into code constructs.

### 2.1. Namespaces and Class Description

In our continuous effort to provide quality services and features, we've made some essential architectural changes in the latest version of TierDeveloper which are very important for mid to enterprise level developments. A separation has been provided between the Domain Objects and the Persistence Layer. Besides this separation, a new Integration Layer has been provided for client interaction as well as a separate assembly has been added for the Custom Hooks. The structure of generated code is slightly different for .NET 1.1 and 2.0. Namespaces generated by the TierDeveloper and their corresponding Classes are discussed and shown in the diagram below:

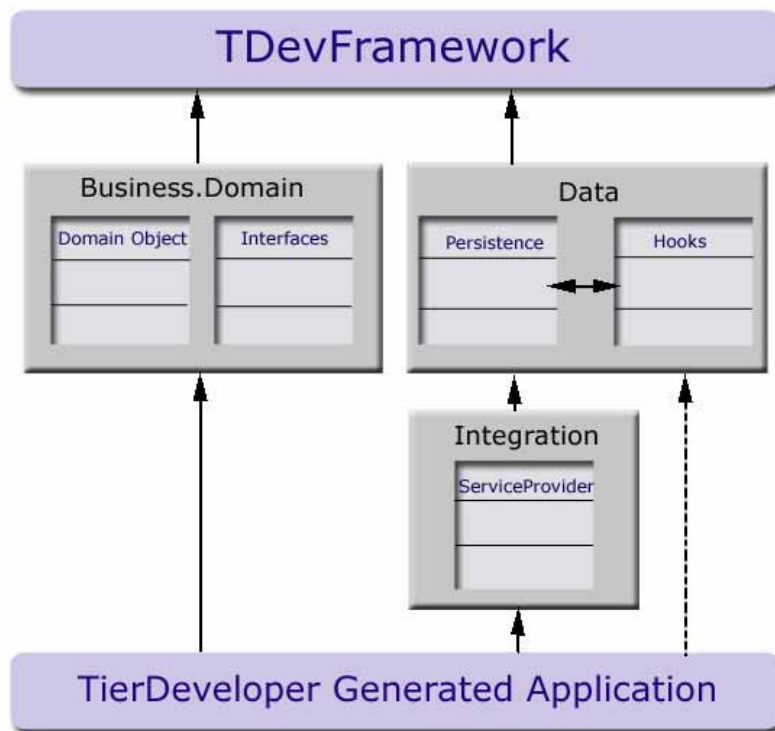


Figure 1: Namespace hierarchy of TierDeveloper generated components

### 2.1.1. Business Namespace

Business namespace is subdivided into Business.Domain and Business.Interfaces. Class view of this namespace of some sample application is shown in figure 2. Business.Domain consists of following classes:

#### Domain Object Class

Domain Object Class represents an entity and determines the state of that entity. This class maps to the relational database table and contains all the attributes of relational table and accessor methods. Domain Object Class is inherited from PersistentObjects Class, which is present in TierDeveloper Framework Library.

#### Collection Class

This class is generated for .NET 1.1 only. For .NET 2.0, Generic Collection "TDBindingList" is supported, that is a part of TDevFramework. Object Collection Class is an extension of the Collection Class, which is present in the TierDeveloper Framework Library. It implements IBindingList and ITypeedList. Besides the methods for adding or removing an object, Sorting, Indexing and Binding are handled by this Class.

#### DS Class

This class inherits directly form DataSet Class. Such a class is called Typed Dataset. Typed DataSet ensures that our DataSets are type-safe when we write the code. Typed DataSets generate classes that expose each object in a DataSet in a type-safe manner.

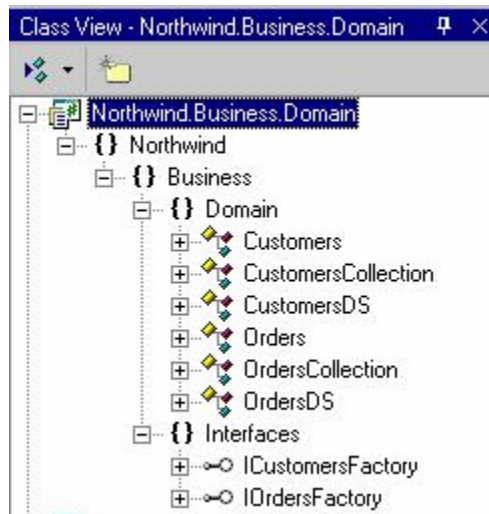


Figure 2: Class view of Business.Domain

Business.Interfaces consists of following class:

#### Interfaces Class

This class provides Interfaces to Factory and Hooks Classes. These Interfaces provide definitions to all transactional methods of an Object. Factory and Hooks classes are responsible to provide implementation of these interfaces.

### 2.1.2. Data.Persistence Namespace

This namespace decouples the domain objects from persistence logic and contains the Factory Classes. *Figure 3* demonstrates the class view of Data.Persistence of some sample application. The description of class is as follows.

#### Factory Class

Factory Class is inherited from DB Object Class, which is present in the TierDeveloper Framework Library. It is a stateless class and is fully transactional. For every object, TierDeveloper generates a separate Factory Class responsible for object persistence. Standard methods such as Create, Read, Update, and Delete are defined in this class. Besides this, all the Custom Operations, Bulk Operations, Stored Procedures and Search Methods are defined here as well. You can do parent-child operations, bulk operations etc., all in single transaction.

#### ServiceProvider Class

This Class provides methods that return new objects of Factory and Hooks Classes.

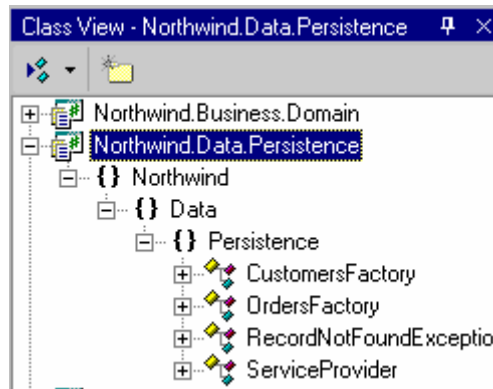


Figure 3: Class view of Data.Persistence

### 2.1.3. Data.Hooks Namespace

This namespace allows you to embed your own code in the application which remains in a separate assembly. We will discuss Hooks in detail later in this document.

*Figure 4* demonstrates the class view of Data.Hooks of some sample application showing all the method definition custom hooks. The description of class is as follows.

#### Hooks Class

This class contains all the Pre Hooks and Post Hooks methods that you specify through TierDeveloper. You are required to provide the implementation yourself. Pre Hooks are called before initiating a database operation while Post Hooks are called after completing a database operation i.e. after calling the methods from the factory class.

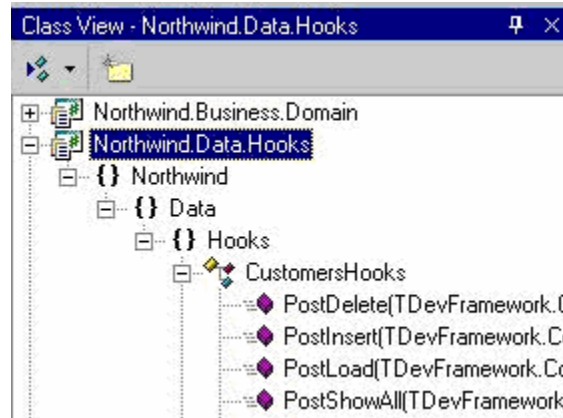


Figure 4: Class view of data hooks

### 2.1.4. Integration Namespace

Client Application talks to Business.Domain, Data.Persistence and Data.Hooks Namespaces via Integration Namespace. Integration layer thus acts as a communication channel between them.

Figure 5 demonstrates the class view of Integration namespace of some sample application.

### ServiceProvider Class

This class contains logic to act as a communication layer between Client Application and Factory Logic.

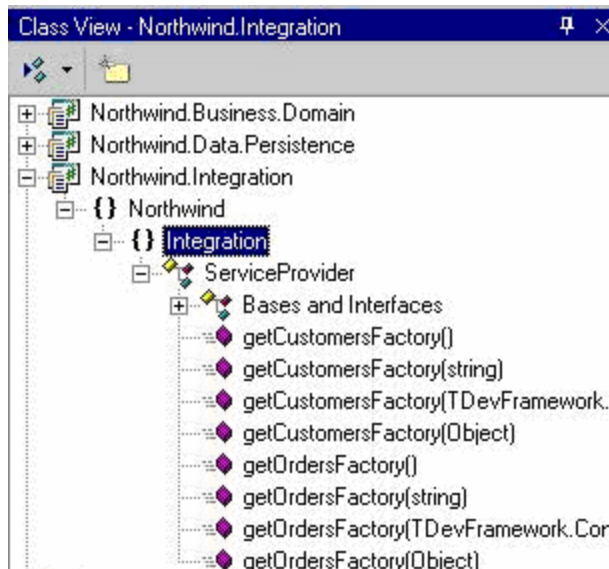


Figure 5: Class view of Integration namespace

### 2.1.5. Webservices

TierDeveloper generates two entities for a WebService, namely Proxy and Server. A client application communicates with a Server through the Proxy. Separate .NET assemblies are generated for both Proxy and Server. TierDeveloper also generates a

Remote Winform Application for testing the functionality of the WebServices.

### **2.1.6. .NET Remoting**

TierDeveloper generates Proxy and Server entities for .NET Remoting. A client can invoke a method on a remote server by using proxy. TierDeveloper generates separate .NET assemblies for both Proxy and Server.

### **2.1.7. Client Applications**

TierDeveloper generates WinForm, ASP.NET, WebServices and Remoting applications and deploys them automatically. Separate assembly is generated for each type of application.

## **2.2. TDevFramework Library**

TierDeveloper provides "TDevFramework" library as part of the generated components code. The basic purpose of this library is to manage the commonly reusable code through a single point of reference. Thus, it eliminates any code duplication across components. All the base classes thus are part of the TDevFramework library. This library should be deployed along with the generated components on the targeted platform. To learn more about TDevFramework Library please read TDevFramework API Reference guide.

## **3. Working with TierDeveloper Components**

The following section outlines the common usage scenarios for the generated components. These demonstrate how a generated component and its methods can be accessed from a (Windows or Web) client application.

The code snippets given here are based on the "Northwind CustomersOrder" Application for which .NET component are generated by TierDeveloper. This is assumed that, the generated source code against the "Customers" and "Orders" Objects consist of all features provided by TierDeveloper i.e. the user has selected all features while defining the Object behavior in TierDeveloper.

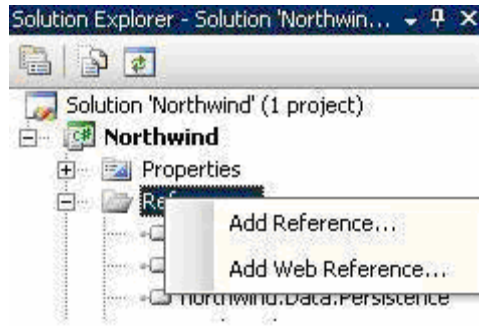
You must add required assembly references in your client application before using the generated components.

### **3.1 Using Generated Components**

The following steps will help you adding the required references.

#### **Step 1:**

Go to the Solution Explorer of Northwind application and right click on the References node as shown in the figure below



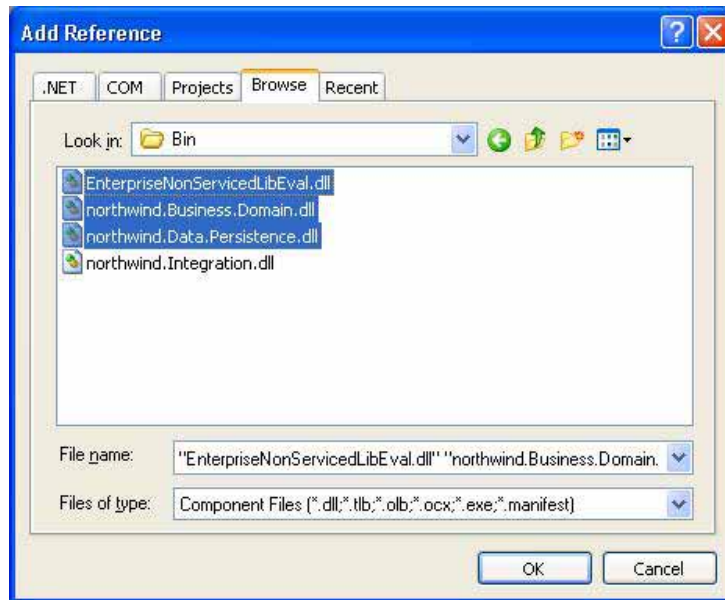
**Figure 6:** Add Reference

**Step 2:**

Browse the TierDeveloper generated assemblies from Data->Persistence->Bin folder of your project and select the following files.

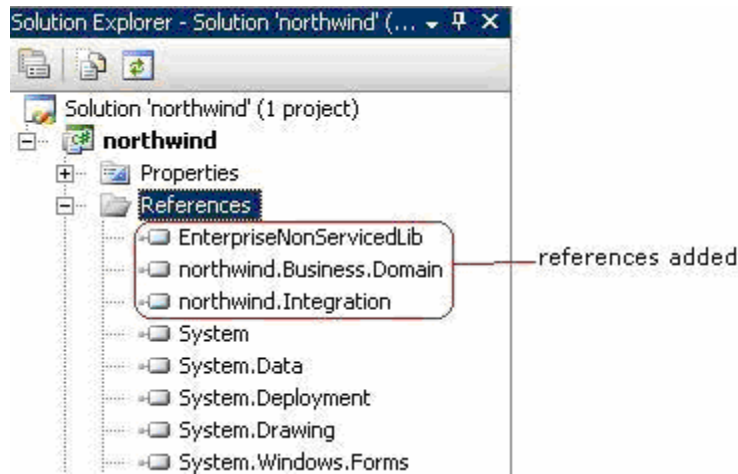
- Northwind.Business.Domain.dll
- Northwind.Integration.dll
- EnterpriseServiceLib.dll

Click **OK** button to continue.



**Figure 7:** Add Reference Dialog Box

Make sure that all the references are correctly added under references node in solution explorer as shown in figure 8.



**Figure 8:** Added references in project

Now you should be able to use the generated components in your application. Please note that 'EnterpriseNonServicedLib' contains base classes defined in TDevFramework Library and 'EnterpriseNonServicedLibEval' is only required during evaluation period.

## 3.2 Class Usage

TierDeveloper generates following three classes for each object:

### 3.2.1. Domain Object Class

Domain objects are the core of any enterprise application which moves between different sub-systems of an application. So for a good design they have to be separated from the persistence logic. Keeping this in view, TierDeveloper generated domain objects contains only getter/setter methods for each attribute and does not contain any persistence logic. Here is how you can use the domain object and its attributes:

[C#]

```
// instantiate an empty domain object, in this case a new Customer
Customers objInfo = new Customers();

// Assign some values to attributes of domain object
objInfo.CustomerID = "CDFNG";
objInfo.ContactName = "Simpson";
objInfo.ContactTitle = "Sales Manager";

// write code here to access any of the auto-generated methods of
// CustomersFactory to perform some operation on the domain object or
// just print them
```

### 3.2.2. Factory Class

Factory is a stateless class that contains all persistence logic. This class derives from DbObject and passes connection through constructor which is then kept in the base class for further use by other methods. Here is the code generated by TierDeveloper to handle connection string through constructor:

```
// default implementation
public CustomersFactory() :
    base(TDevFramework.ConfigurationSettings.AppSettings
        ["Object.Customers"], TDevFramework.EDBProvider.
        SQLCLIENT)
```

```
// Constructor accepting connection string
public CustomersFactory(String connectionString) :
    base(connectionString, TDevFramework.EDBProvider.OLEDB)
```

```
// Constructor accepting connection object
public CustomersFactory(object connection) : base(connection,
    TDevFramework.EDBProvider.OLEDB)
```

```
// Constructor accepting TDevFramework Connection object
public CustomersFactory(TDevFramework.Connection connection):
    base(connection)
```

You can access this class directly or indirectly (through Interfaces) to perform any database operation on Domain Objects. To get a <Object>Factory Class object, in this case CustomersFactory object, write the following code:

[C#]

```
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();
```

[Visual Basic]

```
Dim objFactory As ICustomersFactory =
    ServiceProvider.getCustomersFactory
```

There are four overloaded constructors of a Factory class as mentioned above and are discussed in detail later under "Managing Connection Strings". In the above example, getCustomerFactory() returns an instance of CustomersFactory class. Now you can use this CustomersFactory object to access any transactional code, i.e. Standard Methods, Query Methods, Operations, or Stored Procedures etc. For example, you can call Load method (a single row method) that Loads records of a customer from database on the basis of primary key value.

[C#], [Visual Basic]

```
objInfo.CustomerID = "ANTON";
objFactory.Load(ref objInfo, 0);
```

### 3.2.3. Collection Class

#### For .NET 1.1

Collection class is available for .NET 1.1 only. Collection class is an extension of .Net CollectionBase Class. Its working is somewhat similar to DataSet but it operates on objects instead of rows and columns. It lets you sort the entire collection based on any attribute and provides list management functionality. All query and search methods (methods other than single row operations) return <Object>Collection for .NET 1.1. In the example below, ShowAll query method loads the list of Customers records from the underlying database table and populates the "CustomersCollection" with records.

[Visual Basic]

```
Dim objFactory As ICustomersFactory =
    ServiceProvider.getCustomersFactory

'Instantiate empty collection object
Dim csColl As CustomersCollection = New CustomersCollection

'Add new customers to collection
csColl.Add(New Customers)
csColl.Add(New Customers)

'Save the collection
objFactory.Save(csColl, 0)
```

#### For .NET 2.0

For .NET 2.0, generic collection 'TDBindingList' is provided. TDBindingList Class is a part of TDevFramework. It is derived from System.ComponentModel.BindingList. All query and search methods (methods other than single row operations) return TDBindingList for .NET 2.0.

[C#]

```
TDBindingList<Customers> csColl = new TDBindingList<Customers>();
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();

'Add new customers to collection
csColl.Add(new Customers());
csColl.Add(new Customers());

'Save the collection
objFactory.Save(csColl, 0);
```

### 3.3. Standard Methods

All standard methods, Load, Insert, Update or Delete, are single row operations i.e. they are based on primary key value. All standard methods accept two parameters. One parameter is Domain Object (in our examples it is 'Customers'), and an integer "nDepth". If "nDepth" is greater than '0', standard methods perform operations on

the related entities (i.e. Parent-Child objects) till the specified depth (inclusive). If "nDepth" is less than 0, standard methods perform same operation on all related entities (inclusive), and if it is equal to 0 then the operation is only performed on the object itself.

### 3.3.1. Load

It loads a record from the database based on the PK attribute value of Domain Object and populates the Object. See 'Domain Object Class' and 'Factory Class' under heading 'Class Usage', to learn more about Load method and various attributes of Domain Object.

### 3.3.2. Insert

It creates a new record in the database. In case of auto number (sequence), the Insert method automatically populates the primary key value from the database after successfully completing the operation.

[C#]

```
// define a new customer using Domain Object class Customers
Customers objInfo = new Customers();
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();

// Populate the Object
objInfo.CustomerID = "ANTON";
objInfo.CompanyName = "Antonio Moreno Taquería";
objInfo.ContactName = "Antonio Moreno";
objInfo.ContactTitle = "Owner";
objInfo.City = "México D.F.";

// Now you can insert it as a new record
objFactory.Insert(ref objInfo, 0);
```

### 3.3.3. Update

It updates an existing record in the database based on the primary key attributed.

[C#]

```
// write code here to define a new customer

// give CustomerID of customer whose record to be updated
objInfo.CustomerID = "ANTON";
objInfo.EmailAddr = "neweMailAddress@hotmail.com";
// write code here to Populate Customer object with new values

// update existing record
objFactory.Update(ref objInfo, 0);
```

### 3.3.4. Delete

Delete method requires Primary Key value to delete a record from database, as shown below:

[C#]

```
objInfo.CustomerID = "ANTON";

// Delete record of customer with CustomerID "ANTON"
objFactory.Delete(ref objInfo,0);
```

### 3.3.5. Save

Save method is used to perform bulk operation on generic collection of objects. It takes generic collection 'TDBindingList' as parameter and performs operations based on the state of the each object. Object state can be determined by two flags, i.e. Dirty and IsNew. If 'Dirty' is 'true', Save method updates while if 'IsNew' is 'true', it Inserts a new record into database. You can also specify 'nDepth' to perform the operation up to 'n' hierarchy.

[C#]

```
// Instantiate empty TDBindingList object of Customers
TDBindingList<Customers> custList = new TDBindingList<Customers> ();

// Get a new Factory object, in this case new CustomersFactory
ICustomersFactory custFactory =ServiceProvider.getCustomersFactory();

// Get all Customers in the TDBindingList
custList = custFactory.ShowAll();

// Add a new Customer in TDBindingList
custList.Add(new Customers());

// Save TDBindingList
custFactory.Save(custList, 0);
```

### 3.3.6. Delete Collection

Like Save() method DeleteColl() also performs bulk operation on the given generic collection. It deletes the entire list from the database except the items added with IsNew Status. It also removes the items which are in MarkDeletedItems list.

## 3.4. Queries

The most common thing that a database application does is to retrieve rows of data from one or more tables. The application gets this done by using SQL queries (SELECT statements). However, an object-oriented application wants to fetch a collection of objects and not rows. So, the TierDeveloper provides a way for you to create Query Methods that return generic collections 'TDBindingList' of objects. You can define a query as Count Query, Distinct Query, Partial Query, Parameterized Query or Dynamic Query at the same time. TierDeveloper generates separate Query Methods, does some changes in the SQL statement of the same Query Method, or add some parameters for the same Query Method to provide these features.

### 3.4.1. Count Query

TierDeveloper generates <Query>Count Method for the Query Method. For example, in addition to ShowAll Query Method, a ShowAllCount Query Method will also be generated, that returns number of records found against the query.

```
public Int32 ShowAllCount()
```

[Visual Basic]

```
Dim objFactory As ICustomersFactory =  
    ServiceProvider.getCustomersFactory()  
  
Dim rCount As Integer = objFactory.ShowAllCount()  
Me.lblMessage.Text = "No of record(s) found: " + rCount.ToString()
```

### 3.4.2. Distinct Query

You can get Distinct Records from the database through Distinct Query feature of the TierDeveloper. It places DISTINCT key word in the generated SQL statement for Query Method. For example, Distinct Query for ShowAll Query Method will result in change of SQL statement of the same method and no new method is generated.

### 3.4.3. Partial Query

TierDeveloper allows you to define partial query methods to improve performance of application. It generates an additional <Query>PR Query Method for the Query Method for which you specify to get Partial Data. For example, for ShowAll Query Method, ShowAllPR Query Method is also generated. Partial Query Methods takes two Integer parameters, one is Row position from where to start fetching records, and other is number of records to be fetched and returns the result in a collection.

```
public TDBindingList<Customers> ShowAllPR (Int32 nSP, Int32 nRecords)
```

### 3.4.4. Parameterized Query

A parameterized query returns data that meets the conditions of a WHERE clause. You add parameters to a query by completing the Search Criteria Builder Dialog Box in TierDeveloper. For example, you can parameterize a query to display only customers in a certain city by adding WHERE City = @City to the end of the SQL statement that returns a list of customers. Suppose you have defined a Query Method FindByCity in TierDeveloper that executes parameterized query to give Customers of a specific city. Code generated by TierDeveloper to fetch records will look like as follows:

[C#]

```
TDBindingList<Customers> objList = new TDBindingList<Customers>();  
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();  
  
// search criteria  
string City = "California";  
  
// Get all customers of 'California'  
objList = objFactory.FindByCity(City);
```

### 3.4.5. Dynamic Query

One of the powerful features of TierDeveloper is the support of dynamic queries that takes search criteria and OrderBy clause as run time parameters. Thus we can run the same query with differing filters. So, for example, if you have a dynamic Query

named ShowAll, you can pass "where customer\_id = ALFKI" and "Order by ContactName Ascending" as runtime parameter. Such a feature is handy when you would like to have the ability to run the same query based on different optional criteria for example, Ad hoc Search. If you define ShowAll Query to be a dynamic one, then Query Method will look like the one shown below:

```
public TDBindingList<Customers> ShowAll(String where,String orderBy )
```

Following example shows utilization of dynamic query into client application:

[Visual Basic]

```
Dim strWhere As String = "CustomerID = ALFKI And City = 'California'"
Dim strOrderBy As String = "ContactName Ascending"
Dim myArray As ArrayList
Dim objFactory As ICustomersFactory =
    ServiceProvider.getCustomersFactory()

myArray = objFactory.ShowAll(strWhere, strOrderBy).Items
```

### 3.4.6. Joins (Inner/Outer)

A join combines records from two or more tables in a relational database. TierDeveloper lets you generate both inner and outer joins in Structured Query Language (SQL). You are not required to write even a single line of statement, TierDeveloper does it according to your requirement.

#### Inner Joins

An inner join essentially finds the intersection between the two tables. This is the most common type of join used, and is considered the default join type. In NorthWind database, there is a one-to-many relationship between Customers and Orders and a similar relationship between Orders and Order\_Details. Following example shows, how we can get OrderDetails of a Customer. TierDeveloper generates a query involving complex joins to achieve the task as shown:

```
SELECT
    a31.OrderID OrderID, a31.ProductID ProductID,
    a31.UnitPrice UnitPrice, a31.Quantity Quantity,
    a31.Discount Discount
FROM
    dbo.[Order Details] a31
    INNER JOIN  dbo.Orders a1 ON
        a31.OrderID = a1.OrderID
    INNER JOIN  dbo.Customers a2 ON
        a1.CustomerID = a2.CustomerID
WHERE (a1.CustomerID = ?)
```

Suppose you have defined OrderDetailsByCustomer Query Method in TierDeveloper to execute the above mentioned SQL statement. You can get the results as follows:

[Visual Basic]

```
Dim objList As TDBindingList(Of Orders_Details) = New
    TDBindingList(Of Orders_Details)
Dim objFactory As IOrder_DetailsFactory =
    ServiceProvider.getOrder_DetailsFactory()

string CustomerID = "ANTON"
objList = objFactory.OrderDetailsByCustomer(CustomerID)
```

## Outer Joins (left/right/full)

Through TierDeveloper, you can define query methods with Left Outer Join and Right outer Join. An example of TierDeveloper generated SQL statement having Left join is as follows:

```
SELECT
    a31.OrderID OrderID, a31.ProductID ProductID,
    a31.UnitPrice UnitPrice, a31.Quantity Quantity,
    a31.Discount Discount
FROM
    dbo.[Order Details] a31
    LEFT OUTER JOIN  dbo.Orders a1 ON
        a31.OrderID = a1.OrderID
WHERE (a1.CustomerID = ?)
```

In Right Outer Join, only unmatched rows from the right side table (table-2) are retained. In Full Outer Join, unmatched rows from both tables (table-1 and table-2) are retained.

## 3.5. Stored Procedures

TierDeveloper allows you to boost the performance of your application by using stored procedures. You can generate stored procedures, instead of SQL statements, to be executed by Query Methods. TierDeveloper allows you to put all the SQL (excluding the dynamic queries) inside the DBMS as stored procedures, which is generated as a result of object-relational mapping. In that case, generated objects code (Query Methods, Standard Methods and Operations) is generated to call these stored procedures. To put stored procedures inside DBMS, TierDeveloper generates ".sql" script file, which contains code for generating and placing stored procedures for corresponding Objects. These script files are located in a separate folder named "SpScript". You have to execute these script files using some database client software (SQL query analyzer in case of MSSQL Server or SQLPlus in case of oracle server), in order to use the generated stored procedures.

You can also use existing stored procedures present in your database. TierDeveloper provides a wrapper method around an existing stored procedure so that executing a stored procedure remains similar to calling any other method via an object. A stored procedure has two types of parameters; "In" parameter shows the Input value while "Out" parameter shows the return type of the stored procedure. TierDeveloper also supports return value of type RowSet or collection of objects by a stored procedure.

### 3.4.1. Stored Procedure Returning Rowset

Suppose that the "Customers" Object has a wrapper method called "CustOrderHist" that internally calls a stored procedure named "CustOrderHist". The method populates the "returnValue" ArrayList structure with the rows returned via the stored procedure cursor. Following example shows calling a stored procedure:

[C#]

```
ArrayList myArray = new ArrayList();
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();
String strCustomerID = "ANTON";

// call CustOrderHist wrapper method that executes the stored
// procedure and returns rows in an ArrayList
objFactory.CustOrderHist(strCustomerID, ref myArray);

// Display result in DataGrid
if (myArray.Count > 0)
{
    GridCtrl.DataSource = myArray;
}
```

### 3.4.2. Stored Procedure Returning Collection

Suppose that the "Customers" Object has a wrapper method called "CustbyCity" that internally calls a stored procedure named "CustbyCity". This method populates the Customers collection 'TDBindingList'. Following example shows calling this stored procedure returning collection:

[C#]

```
TDBindingList<Customers> custList = new TDBindingList<Customers> ();
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();
String crtCity = "México D.F.";

// call CustbyCity wrapper method that executes the stored
// procedure and returns rows in TDBindingList of Customers
objFactory.CustOrderHist(strCity, ref custList);

// Display result in DataGrid
foreach (Customers custInfo in custList)
{
    Console.WriteLine("CustomerID:" + custInfo.CustomerID);
    Console.WriteLine("Name:" + custInfo.ContactName);
    Console.WriteLine("Title:" + CustInfo.ContactTitle);
}
```

## 3.5. Operations

Operations are single row methods (Insert, Update, and Load) that are dependent on Primary key. Difference between Operations and Standard methods is that, Operation methods can perform operations on partial set of attributes, thus contribute in overall performance of the application. Suppose that you have defined a

Load Operation "loadPersonalInfo" on Customers Object, in TierDeveloper, that loads a record of Customer's Personal Information from the database and populates the Customers object with the record attributes. Example shows how you can utilize that Operation in your application.

[C#]

```
// Instantiate an empty domain object, in this case a new Customer
Customers objInfo = new Customers();

// Get CustomersFactory Object
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();

// Give CustomerID of Customer whose record to be Loaded
objInfo.CustomerID = "ANTON";
objFactory.loadPersonalInfo(ref objInfo);

// populate Form's Labels with the loaded attributes
this.lblContactName.Text = objInfo.ContactName.ToString();
this.lblAddress.Text = objInfo.Address.ToString();
this.lblCity.Text = objInfo.City.ToString();
this.lblCountry.Text = objInfo.Country.ToString();
```

[Visual Basic]

```
`Instantiate an empty domain object, in this case a new Customer
Dim objInfo As Customers = New Customers

`Get CustomersFactory Object
Dim objFactory As ICustomersFactory =
    ServiceProvider.getCustomersFactory

`Give CustomerID of Customer whose record to be Loaded
objInfo.CustomerID = "ANTON"
objFactory.loadPersonalInfo(ref objInfo)

`Populate form labels with the loaded attributes
Me.lblContactName.Text= objInfo.ContactName.ToString()
Me.lblAddress.Text= objInfo.Address.ToString()
Me.lblCity.Text= objInfo.City.ToString()
Me.lblCountry.Text= objInfo.Country.ToString()
Me.lblCustomerID.Text= objInfo.CustomerID.ToString()
```

Similarly, you can update partial set of attributes of any object. Suppose you have generated an Operation "UpdatePersonalInfo" that updates personal information of a customer, not the whole record. See how you can use that Operation in your own application:

[C#]

```
// Instantiate an empty domain object, in this case a new Customer
Customers objInfo = new Customers();

// Get CustomersFactory Object
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();

// Populate the Object
objInfo.ContactName = "Antonio Moreno";
objInfo.City = "México D.F.";
objInfo.Country = "Mexico";
objInfo.Address = "Mataderos 2312";

// Give CustomerID of Customer whose record is to be Updated
objInfo.CustomerID = "ANTON";

// Update
objFactory.UpdatePersonalInfo(ref objInfo, 0);
```

## 3.6. Bulk Operations

Bulk Operations work on specific set of attributes in the table for all the records that qualify the given criteria.

### 3.6.1. Bulk Update Operation

Bulk Update Operation updates the specific set of attributes in the table for all the records that qualify the given criteria. Suppose you have generated a Bulk Update Operation “updatePostalCodebyCity” for Customers. Let us see how you call this Operation from your own application.

[C#]

```
// Get CustomersFactory Object
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();

// give name of the city whose PostalCode you want to change
string City = "México D.F.";
string PostalCode = "34526";

// Update Postal code of México D.F.
objFactory.updatePostalCodebyCity(City, PostalCode);
```

### 3.6.2. Bulk Delete Operation

Bulk Delete Operation removes all records from table that qualify the given criteria. Suppose you have generated a bulk Delete operation on Customers, “deleteCustomersbyTitle”, which deletes all Customer records that qualify the ContactTitle criteria. Example shows usage of Bulk delete operation in your code.

[C#]

```
// Get CustomersFactory Object
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();

// give title of the customer whose record you want to delete
string ContactTitle = "Sales manager";

// Delete all Sales Managers
objFactory.deleteCustomersbyTitle(ContactTitle);
```

## 3.7. Relationships

TierDeveloper automatically detects all relationships defined at the database level that includes one-to-one, one-to-many, many-to-one and many-to-many relations. Additionally, it lets you define Relationships behavior for the "Custom" relations that are created in the tier project and resides only in the object model (Offline Schema).

The parent Object keeps a reference to all its related objects and manages insertion, updating and deletion of child object(s). All these parent-child hierarchical operations are performed in single transaction and if any of the operation fails the whole transaction rolls back.

### 3.7.1. One-to-One / Many-to-One Relationship

For handling one-to-one relationship, TierDeveloper generates code in which object contains a reference to other related objects and handles load and save scenarios for them. Code generated for handling many-to-one relationship is similar to the code for one-to-one relationship. The following code illustrates one-to-one relationship between Person and Customer class.

[C#]

```
// Instantiate an empty Person object
Person personInfo = new Person();

// Get PersonFactory Object
IPersonFactory objFactory = ServiceProvider.getPersonFactory();

// Give PersonID of Person whose record to be Inserted
personInfo.PersonID = "ASRTD";

// Instantiate the Customer object
personInfo.Customer = new Customers();
// populate Customers Object that resides in Person
personInfo.Customer.CustomerID = "AGTFD";
personInfo.Customer.ContactName = "John";
personInfo.Customer.City = "California";

// Insert Person, 1 depth specifies Insertion of Customer record too
objFactory.Insert(ref personInfo, 1);
```

### 3.7.2. One-to-Many Relationship

For one-to-many relationship, object contains a collection of the related object. As you know in Northwind database there is a one-to-many relationship between Customers and Orders tables. In that case Customer Object will contain a collection of Orders. Let us look at the following example to learn how easy it is to perform

parent-child operations. The following code will load the customer object along with all associated orders. Here the parameter nDepth is used for deep operation.

```
[C#]
// Instantiate an empty Customer object
Customers objInfo = new Customers();

// Get CustomersFactory Object
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();

// Give CustomerID of Customer whose record to be loaded
objInfo.CustomerID = "ANTON";

// Load records of Customer, 1 depth specifies loading of Orders
// record too
objFactory.Load(ref objInfo, 1);

// Get All Orders of the Customer with CustomerID "ANTON"
foreach (Orders OrderInfo in objInfo.Orders)
{
    Console.WriteLine("Order ID:" + OrderInfo.OrderID);
    Console.WriteLine("Order Date:" + OrderInfo.OrderDate);
    Console.WriteLine("Required Date:" + OrderInfo.RequiredDate);
    Console.WriteLine("Shipped Date:" + OrderInfo.ShippedDate);
}
```

### 3.7.3. Many-to-Many Relationship

TierDeveloper maps many to many relationships in your objects in a somewhat efficient manner. Let us first look at what does a many-to-many relationship look like in the database. Suppose you have a many-to-many relationship between Products and Orders tables via a bridge table called OrderDetails. The bridge table's primary key consists of two foreign keys coming from each of the corresponding tables. Additionally, the bridge table has additional attributes for the many-to-many relationship itself. Now many-to-many relationship between Products and orders looks like two relationships, one is one-to-many between Products and OrderDetails and other is many-to-one between OrderDetails and Orders. In that case, to have a consistent design, TierDeveloper generates Products, Orders, and Orderdetails objects so that Orders and Products table both keep a collection of OrderDetails objects.

Now let us see how the persistence code looks. In the Load method of ProductsFactory, the Product object loads from the database in a normal fashion, returning all attributes of the Products and Collection of OrderDetails (see one-to-many relationship above). And each OrderDetails object also points to its related (n-1) Orders object. To fully understand how TieDeveloper manages many-to-many relationships please read "Many-to-Many Relationships in O/R Mapping".

## 3.8. Managing Connection String

### 3.8.1 Dynamic Connection

It is always a good practice to keep your data source and other related information in a configuration file because it makes the maintenance easy. Keeping this in view, TierDeveloper stores data source information in a configuration file. You can specify different connection strings for individual objects in this Configuration File. Following text in the Configuration File sets the connection string for the application:

```
<configuration>
  <appSettings>

    <add key="Ds.Northwind" value="Provider=SQLOLEDB;Data
Source=localhost;Database=Northwind;User Id=sa;Password=;" />

    <add key="Object.Customers" value="Provider=SQLOLEDB;Data
Source=localhost;Database=Northwind;User Id=sa;Password=;" />

    <add key="Object.Orders" value="Provider=SQLOLEDB;Data
Source=localhost;Database=Northwind;User Id=sa;Password=;" />

  </appSettings>
</configuration>
```

ConfigurationSettings is a utility class in TDevFramework Library that parses the Configuration File and reads its properties. Now, for example, you can read the value of the Object.Customers (Connection string for Customers) parameter like this:

```
TDevFramework.ConfigurationSettings.AppSettings[ "Object.Customers" ])
```

By default factory classes use connection string specified in Configuration File. However, you can also specify custom connection string at run time.

```
public static ICustomersFactory getCustomersFactory()
```

### 3.8.2 Custom Connection

In some situations you may need to change the connection string at run time and for that purpose TierDeveloper generate overloaded factory constructors. For example to support transaction between unrelated objects you may need to specify custom connection string to factory classes at runtime.

To specify a custom connection you can pass a connection string to getCustomersFactory.

```
public static ICustomersFactory getCustomersFactory(string strConn)
```

You can also pass a TDevFramework.Connection object to getCustomersFactory.

```
public static ICustomersFactory  
    getCustomersFactory(TDevFramework.Connection tdevConn)
```

Custom connection can also be Connection Object of any of the data providers such as SqlConnection, OleDbConnection etc.

```
public static ICustomersFactory getCustomersFactory(Object objConn)
```

### 3.9 Support for Nullable types

This feature is supported in .NET 2.0 only. A null value in a relational database is used when the value in a column is unknown or missing. TierDeveloper provides IsNull and SetNull methods in this context. This is particularly important when you want to insert null DateTime value into database, otherwise vague Date is inserted into database.

IsNull method checks whether given field is null or not.

```
public bool IsNull(string szFieldName)
```

SetNull method sets the field value Null or NotNull.

```
public void SetNull(string szFieldName , bool bNull)
```

Given these methods, you can check whether a field value is null after fetching records from database, and can also set a field as Null before inserting it into database.

[C#]

```
// define a new customer using Domain Object class Customers  
Customers objInfo = new Customers();  
  
// Set Country as Null.  
objInfo.SetNull(objInfo.Country, true);  
  
// Perform some operation on Customers object
```

In its version 5.6, TierDeveloper has added Nullable support provided by .Net framework itself. Following is the code snippet, generated through TierDeveloper, which will give you an idea of how it can be done using .Net Nullable technique.

```
[C#]
public Int32? age
{
    get
    {
        if (this["age"].Null)
            return null;
        return (Int32?) this["age"].Value;
    }
    set
    {
        this["age"].Value = value;
    }
}
```

In the above example, "age" of an "Employee" is declared as Nullable integer. Integers couldn't traditionally be nulls (or Nothing) because they were value types and value types are not references like strings. Nullable type is constructed from the generic Nullable structure. By having this feature, you can submit null values to your database table field normally, like submitting a null DateTime value to a datetime field in your database table. You can also check if your nullable value type contains data. Following code Loads an Employee data and checks whether age is available or not:

```
[C#]
Employee objInfo = new Employee();
IEmployeeFactory objFactory = ServiceProvider.getEmployeeFactory();

objInfo.EmployeeID = 1;
objFactory.Load(ref objInfo, 0)

If (objInfo.age == null)
{
    Console.WriteLine("age is Null");
}
```

Or check for value through HasValue property attached to your nullable value type

```
[C#]
If (objInfo.age.HasValue)
{
    //do something
}
```

The HasValue property returns true if the variable contains a value, or false if it is null. The Value property returns a value if one is assigned, otherwise a [System.InvalidOperationException](#) is thrown.

### 3.10 DataSets

TierDeveloper supports generation of both Typed and Untyped DataSet Interfaces for Query Methods. As we have discussed before, Query Methods and Operations that

fetch rows from database, populate Object Collection 'TDBindingList' and return that Collection. When Dataset interfaces are generated, additional Query Methods and Operations are generated that return DatSets. These methods have the same name as the Query Methods with a 'DS' or 'TDS' suffix, showing return type as DataSet or Typed DataSet respectively. For example, for ShowAll Query Method, ShowAllDS and ShowAllTDS Methods are also generated.

### 3.10.1 Updateable DataSets

Advantage of using DataSets is that we can bind a DataSet to UI controls. The Dataset then can be updated to save changes made by the user (if any). TierDeveloper provides method to update dataset thus makes the whole process very easy. Suppose you have fetched all Customers by calling ShowAllDS method and bind the result with data grid. You also want to update the dataset if user adds, deletes, or changes some data through DataGrid, when you click update\_ds button. This is how the code will look like:

[Visual Basic]

```
Dim objFactory As ICustomersFactory
Dim ds As DataSet

'Bind DataSet to Data Grid Control
Private Sub CustomersForm_Load(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

    objFactory = ServiceProvider.getCustomersFactory
    ds = objFactory.ShowAllDS()
    DataGrid1.DataSource = ds

End Sub

'Update DataSet
Private Sub update_DS_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles update_DS.Click

    objFactory.UpdateShowAllDS(ds)

End Sub
```

### 3.10.2 Typed DataSets

TierDeveloper supports Typed Datasets, allowing you to access the tables and columns with user-friendly names and strongly typed variables. A "Typed DataSet" class derives from a DataSet Class, inheriting all the methods, events, and properties of a DataSet. By using Typed Datasets in your code you can access tables and columns by name, instead of using collection-based methods. In addition, any miss match results in compile time error rather than a run time error. Suppose you wish to generate typed DataSet for Customers object. In that case, TierDeveloper generates a class CustomerDS that inherits from System.DataSet Class. The Class Diagram of CustomerDS in TierDeveloper Framework looks like the one shown in figure below:

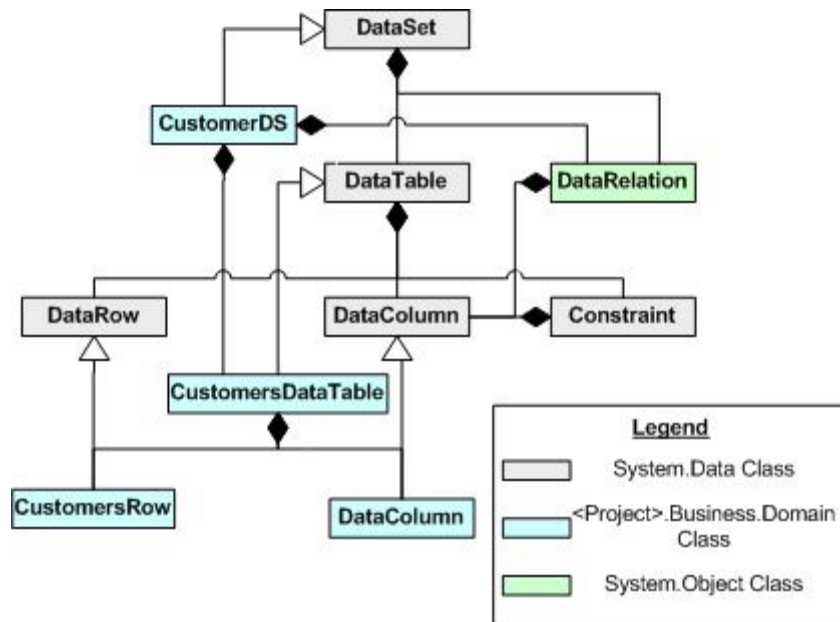


Figure 9: Class diagram of CustomerDS (dataset)

In the example below, you should notice that the syntax to get the tables and fields is much more straightforward with Typed DataSets. Each DataTable is now referenced with a property of CustomerDS. Likewise, each field is a property of a row. Not only is this syntax more straightforward, but you will get compiler errors if you misspell any of the elements.

[Visual Basic]

```

Dim i As Integer
Dim objFactory As ICustomersFactory =
    ServiceProvider.getCustomersFactory()

Dim ds As CustomersDS = objFactory.ShowAllTDS()

'Count Method gives total number of rows, iterate through each row
'and print records
For i = 0 To ds.Customers.Count - 1
{
    Console.WriteLine(ds.Customers(i).CustomerID.ToString())

    Console.WriteLine(ds.Customers(i).ContactName.ToString())

    Console.WriteLine(ds.Customers(i).CompanyName.ToString())
    Console.WriteLine(ds.Customers(i).Address.ToString())
}
Next i

'this will not compile because CustomerID expects a string
ds.Customers(0).CustomerID = 12345

```

## 3.11 Object Binding

Object Binding is a powerful feature provided by the TierDeveloper framework that enables visual elements in a client to connect to a TierDeveloper generated objects. Some of the visual elements in the client can be TextBox, Datagrid, etc. A two-way connection is established such that any changes made to the Objects are reflected immediately in the visual element and vice versa. To support Object binding, both Domain Object and Collection classes (for .NET 1.1 only) implement IBindingList which provides the features required to support both complex and simple bindings. You have to enable the Object Binding support in TierDeveloper.

### 3.11.1 Simple Binding

The ability of a control to bind to a single data element is called Simple Binding, such as a value in a column in a dataset table. This is the type of binding typical for controls such as a TextBox controls or Label controls, which are controls that only displays a single value.

Let's assume you've three textbox controls on a win form and you have loaded all Customers in a generic collection by calling ShowAll method of CustomersFactory class. Here is how you might want to load Customers and then bind the attributes to text controls (in your case you might have data grid or other UI components).

[Visual Basic]

```
Dim objList As TDBindingList(Of Customers)

Private Sub btnLoad_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnLoad.Click

    Dim objCustomer As Customers = New Customers
    Dim objFactory As CustomersFactory = New CustomersFactory
    objList = New TDBindingList(Of Customers)
    objList = objFactory.ShowAll()

    'Bind data to text boxes
    txtCustomerId.DataBindings.Add("Text", objList, "CustomerId")
    txtCompanyName.DataBindings.Add("Text", objList, "CompanyName")
    txtContactTitle.DataBindings.Add("Text", objList, "ContactTitle")

End Sub
```

You might allow users to make changes to any records and then persist those changes when the user clicks SAVE button. Your click event handler for "Save" button might persist the changes with a single call as illustrated below.

[Visual Basic]

```
Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnSave.Click

    Dim cf As CustomersFactory = New CustomersFactory

    'Save changes that you have made in Customers collection
    cf.Save(objList, 0)

End Sub
```

### 3.11.2 Complex Binding

The ability of a control to bind to more than one data element, typically more than one record in a database is called Complex Binding. Complex binding is also called list-based binding. Examples of controls that support complex binding are the DataGridView, ListBox, and ComboBox controls. Below is the example of binding a DataGridView control with TierDeveloper generated collection.

[Visual Basic]

```
Dim objList As TDBindingList(Of Customers)
Dim objFactory As ICustomersFactory =
    ServiceProvider.getCustomersFactory()
objList = objFactory.ShowAll()

'Bind data to DataGridView
Me.DGrid.DataSource = objList
```

Once you bind DataGridView to Collection, you can update the Collection in case of any modification in it through DataGridView, as shown in the previous example.

## 3.12 Code Customization

### 3.12.1 Custom Hooks

You'll always have situations where you need to customize generated code. However, if you change the generated code, it will most likely get overwritten the next time you generate code again. To ensure that your custom code gets called seamlessly, TierDeveloper supports the concept of "Hooks" where you can make use of the "Custom Hooks Skeleton Classes" to incorporate your business logic without making any changes in the generated code. Hooks also enables your custom code to control subsequent execution of the generated code.

TierDeveloper creates a separate assembly for the custom hooks to provide better accessibility and readability of the custom logic. "Pre" and "Post" Hooks are generated for database access logic. "Pre hooks" methods are called before whereas "Post Hook" methods are called after the database operations.

Suppose you want to maintain a Log of Inserted records in the database. For that purpose, you have to keep logic of maintaining Log before and after record insertion. You can generate Pre and Post Hooks for Insert i.e. PreInsert and PostInsert Methods, and place logic to maintain Log. SUCCESS\_CONTINUE,

SUCCESS\_NONCONTINUE, FAIL\_NONCONTINUE, tell database access methods whether to continue executing the access logic after calling Pre and Post Hooks.

```
public const int SUCCESS_CONTINUE = 0;
public const int SUCCESS_NONCONTINUE = 1;
public const int FAIL_NONCONTINUE = 2;
```

For your convenience and better understanding we have listed sample code here which shows how TierDeveloper embed custom hooks in the generated code. It is pretty self-explanatory.

[C#]

```
public int PreInsert(TDevFramework.Connection Conn,Customers objInfo)
{
    // put your own code here
    return SUCCESS_CONTINUE;
}

public int PostInsert(TDevFramework.Connection Conn,Customers
                                objInfo)
{
    // put your own code here
    return SUCCESS_CONTINUE;
}
```

### 3.12.2 Partial Classes

Using Partial Classes, it is possible to split the definition of a class or an interface over two or more source files. Each source file contains a section of the class definition, and all parts are combined when the application is compiled. Splitting a class definition is desirable when working on large projects. Spreading a class over separate files allows multiple programmers to work on it simultaneously.

TierDeveloper generates Partial Classes for .Net 2.0 components. You can keep the custom code inside these partial classes. You can also regenerate the code without the fear of losing the code inside these partial classes.

### 3.12.3 Template Customization

TierDeveloper is a template based code generation tool. It combines the templates with a combination of object mapping and the database schema information to determine exactly how to generate the code. Since TierDeveloper generates code from templates, it lets you modify these templates (or add new templates) so you can affect the structure of generated code.

A very simple example would be when you would like to put your own copyright header in each source code file. If you could go on inserting this header in the code template file, it would automatically get used next time you generate code. You should also be able to write your own code templates (although this is only for advanced users) and let the O/R mapping tool use your templates but do everything else the same way as it always does.

### 3.13 Lazy Loading

TierDeveloper supports Lazy Loading, which is useful when the Child Object will not always necessarily be used and therefore will not always need to be loaded. Second, since it checks to see if the Object has already been loaded, it will not reload the data multiple times. This is especially important to do performance-wise for variables that may load large amounts of data so that it does not needlessly allocate extra space in memory.

For instance, if we have a Customers root object, which has a collection of child objects (Orders) then we can use lazy loading on the Orders collection. You will notice the code change in the Parent Object (Customers) for the Orders property. Every time you reference this property, it will determine whether to Load the Child Objects or not.

To pursue the Customers/Orders example, in the Customers class we'd have a variable to store the child collection:

```
private TDBindingList<Orders> _Orders = null;
```

We'd also have a property to expose this to clients, where we'd use lazy loading:

[C#]

```
public TDBindingList<Orders> Orders
{
    get
    {
        if(_Orders != null)
        {
            return _Orders;
        }
        else
        {
            if( iFactory != null && !IsNew && !
this["CustomerID"].Null)
            {
                Customers objInfo = this;
                iFactory.LoadOrders(ref objInfo,0);
            }
            else
            {
                _Orders = new TDBindingList<Orders>();
            }
            return _Orders;
        }
    }
    set
    {
        _Orders = value;
    }
}
```

We then need to make sure that we have only loaded the Customer object from the database but not its orders. This requires specifying nDepth to 0.

[C#]

```
Customers objInfo = new Customers();

// Get CustomersFactory Object
ICustomersFactory objFactory = ServiceProvider.getCustomersFactory();

// Give CustomerID of Customer whose record to be loaded
objInfo.CustomerID = "ANTON";

// Load records of Customer, 0 depth specifies that Orders records
// are not loaded and will be loaded on demand
objFactory.Load(ref objInfo, 0);
```

By setting Orders to Null, we ensure that subsequent attempt to access this information will trigger the lazy load operation for example, by accessing objInfo.Orders will load the customer orders. Notice that we didn't load the data here, we just defer the load until sometime in the future (maybe).

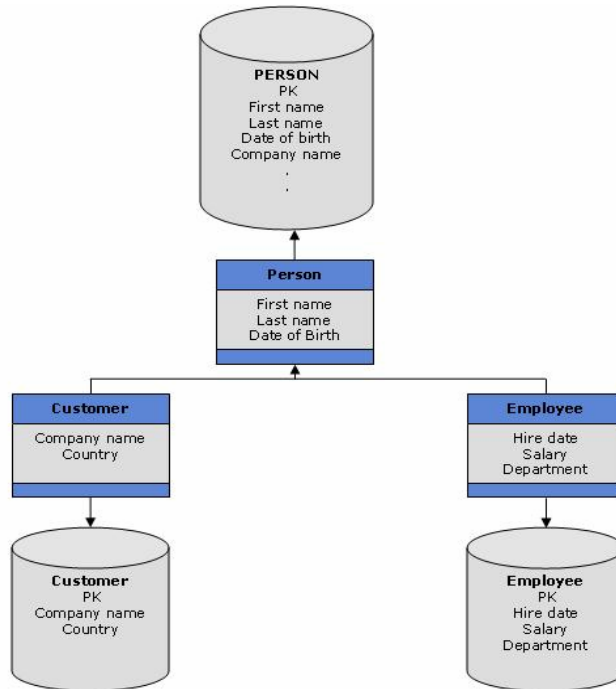
## 3.14 Object Inheritance

Relational databases do not automatically provide inheritance in the relational model. But, there are a number of patterns on how to map object inheritance to a relational database. TierDeveloper provides two approaches for object inheritance.

- Vertical inheritance mapping
- Filtered inheritance mapping

### 3.14.1 Vertical inheritance mapping

In Vertical Inheritance mapping, each class in the inheritance hierarchy maps to its own table in the database. Base and derived tables should be having a one-to-one relationship. After inheritance is defined, the foreign key of this relationship is kept in the derived object. A separate "load" is done for each object. In the generated code, the derived class first asks the base class to do Insert or Update and then do its own. Unlike the insert and update operations, the delete operation are performed first on the derived object and then on the base object. The load operation in the derived class also calls load on the base class. But both the base and derived class operations are performed in one transaction.



For the scenario shown in the figure, Employee Class will be derived from Person Class, as shown in TierDeveloper generated code below:

[C#]

```
public class Employee : Person, IComparable
```

Following example shows that by calling load of derived class object, we will be loading attributes of base class object too, that are accessible through derived class. On the other hand, base class loads its own attributes only.

[C#]

```
Employee empInfo = new Employee();
IEmployeesFactory empfact = ServiceProvider.getEmployeesFactory();

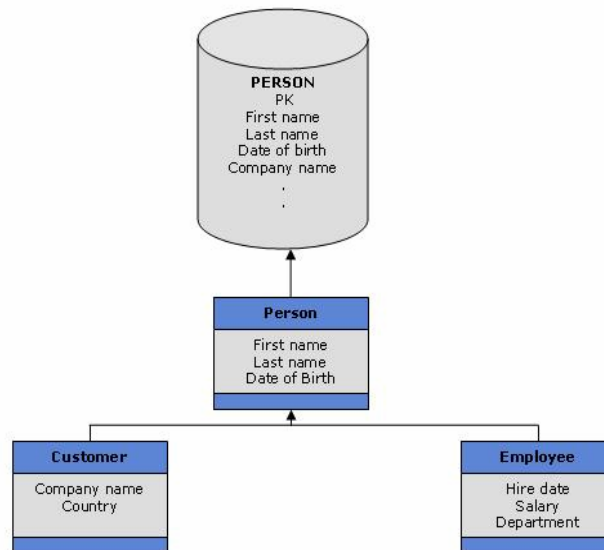
//Employee uses Primary key of Person
empInfo.PersonID = 1;
empfact.Load(ref emp, 0);

//Attributes of Person
Console.WriteLine(empInfo.FirstName);
Console.WriteLine(empInfo.LastName);
Console.WriteLine(empInfo.Age);

//Attributes of Employee
Console.WriteLine(empInfo.Salary);
Console.WriteLine(empInfo.Department);
```

### 3.14.2 Filtered inheritance mapping

In filtered inheritance mapping, persistence subclasses are mapped to one single table which contains all fields of subclasses. A filter column (discriminating key) is included to distinguish between subclasses. Filter inheritance mapping is the fastest of all inheritance models; it never requires a join to retrieve a single persistent instance from the database. Similarly, persisting or updating a single persistent instance requires only a single insert or update statement. Filtered inheritance is best practiced in cases where different roles are utilized by same entity, for example, in any system employee, customer, and person contain data on the base of PersonID. So we use filtered inheritance and map these three classes to single table and defined key field, on the basis of which data is fetched for required functionality.



### 3.15 Transactions

A database transaction allows you to group multiple operations as one atomic operation so either all operations succeed or none of them succeed. Transactional operations include Insert, Load, Update, Delete and Query methods. You might be working with Serviced Components (COM+/MTS) or non-serviced components (Stand-alone), and both types have two different transactional environments. TierDeveloper knows both of them and generate the code accordingly. TierDeveloper supports all MTS transaction types such as Required, RequiresNew, Disabled, Supported and NotSupported for serviced components. Similarly, different isolation levels such as ReadCommitted, ReadUncommitted, RepeatableRead, Serializable are also supported for non-serviced components.

#### 3.15.1 With Related Objects

Transactions in parent-child relationships are handled automatically by the generated code. In case of a Customers, Orders, and OrderDetails relationship hierarchy, all database operations are performed in single transaction. In case of any error, whole transaction rolls back. We have discussed Operations on Child Objects in standard operations before. In the following code snippet you can see how our Load operation loads Customers and its associated Orders in one transaction:

[C#]

```
public void Load(ref Customers objInfo, int nDepth)
{
    try
    {
        // Begin transaction by passing isolation level or none
        BeginTransaction();

        // Add command parameters and execute reader to Load
        // Parents record here

        if (nDepth != 0)
        {
            // Load Orders record here
            LoadOrders(ref objInfo, nDepth);
        }
        // Commit transaction
        SetStatus(EStatus.eSuccess);
    }
    catch (Exception e)
    {
        // Roll back transaction
        SetStatus(EStatus.eFail);
    }
}
```

### 3.15.2 With Unrelated Objects

There comes situation when you want to perform operations on unrelated objects in one transaction. For example, you may want to transfer some amount from one account to another. Accounts might be unrelated objects, but you have to deduct and add amount from both accounts in one go. These objects are may be serviced or non-serviced, depending upon how you define them in TierDeveloper. In both cases, transactions are handled differently.

#### 3.15.2.1 For Serviced Components

In case of serviced components, Microsoft Transaction Server (MTS) manages all transactions of an application. Your objects do not start, commit, or rollback a transaction. They only return success or failure from their methods and MTS figures out when to do "BeginTrans", "Commit", or "Rollback". Additionally, all your factory objects are stateless so MTS can do object pooling on them. TierDeveloper understands and generate your persistence objects to comply with it.

To handle the transaction for the above example, we have to have a third class, let's say TransactiontwoInOne, whose transaction type should be "Required" or "RequiresNew". Then we have to call factory methods of both account objects from within TransactiontwoInOne. Make it sure that transaction types of both account objects is "Required" not "RequiresNew". By doing this, both accounts will join transaction started for TransactiontwoInOne and all methods will execute in a single transaction. Pseudo code for the example would be as follows:

[C#]

```
// Specify Transaction type "RequiresNew" for this class here
[TransactionType(TransactionOption.RequiresNew)]
Class TransactionTwoInOne
{
    void TransferAmount ()
    {
        IAccount1Factory act1Fact =
            ServiceProvider.getAccount1Factory();
        IAccount2Factory act2Fact =
            ServiceProvider.getAccount2Factory();

        // amount to be transferred
        double amount= 10,000;

        // Account number from where to deduct
        string act1Number= "1258-7";

        // Account number where to add amount
        string act2Number= "4558-2";

        // Perform the transfer in single transaction using MTS
        act1Fact.DeductBalance(act1Number,amount);
        act2 Fact.AddBalance(act2Number,amount);
    }
}
```

### 3.15.2.2 For Non-Serviced Components

When not specifically specified, TierDeveloper generates non-serviced components for your application. This is the environment where your application manages all the transactions itself. It needs to know where to go "BeginTrans", "Commit", and "Rollback". TierDeveloper is aware of this environment and generates code to comply with it. See the example below to learn how to handle transaction for unrelated non-serviced objects:

[C#]

```
// make a new TDevFramework connection
TDevFramework.Connection conn = new TDevFramework.Connection();
conn.OpenConnection();

// Instantiate Factory objects of unrelated components
// with Connection object
IAccount1Factory act1Fact = ServiceProvider.getAccount1Factory(conn);
IAccount2Factory act2Fact = ServiceProvider.getAccount2Factory(conn);

// Begin transaction
conn.BeginTransaction();

// amount to be transferred
double amount= 10,000;

// Account number from where to deduct
string act1Number= "1258-7";

// Account number where to add amount
string act2Number= "4558-2";

// Perform the transfer in single transaction
act1Fact.DeductBalance(act1Number,amount);
act2Fact.AddBalance(act2Number,amount);

// Commit transaction
conn.SetStatus(TDevFramework.EStatus.eSuccess);
```

### 3.16 Web Services

Web Services provide the most flexible and standardized infrastructure for creating distributed computing applications that was not available in other technologies like COM, RMI etc. A web service in its simplest form is an object oriented class residing on a web server, and allowing remote clients to invoke its public interfaces. Considering the importance of web services, TierDeveloper has provided a much easier and convenient way of generating web services. TierDeveloper generates two entities for a WebService, namely Proxy and Server. A client application communicates with a Server through the Proxy. Separate .NET assemblies are generated for both Proxy and Server. You can expose and deploy an object method(s) as web method(s) to your client applications in an easy way. In the example below ShowAll method is defined as a WebMethod in Server, and called through Proxy.

[Visual Basic]

```
Dim myArray As CustomersWS.Customers()
Dim objFactory As CustomersWS.CustomersFactorySrv =
    ServiceProvider.getCustomersFactory

myArray = objFactory.ShowAll()
```

## 3.17 .NET Remoting

NET remoting enables you to build widely distributed applications easily, whether the application components are all on one computer or spread out across the entire world.

When Remoting application is generated using TierDeveloper, a console application 'Remoting.Server.exe' is created along with the generated components. You will find this application at Remoting\Server\bin folder of the application. To open the Channels, used to send/receive messages between applications across remoting boundaries, this application should be running. TierDeveloper provides "TCPChannel" for communication. Following example shows calling a method from client application through proxy:

```
[C#]
Employees empInfo = new Employees();

IEmployeesFactory empfact =
    Remoting.Proxy.ServiceProvider.getEmployeesFactory();
empInfo.PersonID = 1;
empfact.Load(ref emp, 0);
```

In order to run client and server on different machines, you need to update server name in remoting.proxy.dll.config file of client application. Give the server name in place of 'localhost' in remoting URL in 'ClientApp\bin\xxx.remoting.proxy.dll.config' to execute the above mentioned code, when client and server are on two different machines.

```
URL = "tcp://localhost:2252/EmployeeFactory.rem"
```

## 3.18 Serviced Components

TierDeveloper lets you generate .NET serviced components for your application. In the .NET world, COM+ components are referred to as serviced components. Due to the number of changes in the .NET strategy, the deployment model for serviced components is different from deploying traditional COM+ components. COM+ runtime provides services to components such as object Pooling, database Connection Pooling, Resource Sharing, Role Based Security and Distributed Transaction Processing.

### 3.18.1 Dual Interface

COM clients and COM+ services communicate with .NET components using the interfaces provided by the .NET components. These are important in defining how COM and COM+ interact with the managed components. ClassInterface attribute specifies how the interfaces will be generated for a class, if they will be generated at all. If ClassInterface value is set to AutoDual, it automatically creates an interface that exposes all the public members of a class. Secondly, it generates dual interfaces meaning that COM clients can use these interfaces for both late and early binding. All the type information is produced for the class interface and published in the type library. TierDeveloper automatically sets ClassInterface to AutoDual, and creates

interfaces for our components that are visible to COM, by adding following line before class definition:

```
[ClassInterface(ClassInterfaceType.AutoDual)]
```

## 3.19 Ajax Support

TierDeveloper allows you to generate Ajax enabled code for your applications. Ajax (also known as AJAX), shorthand for "Asynchronous JavaScript and XML", is a development technique for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change. This is intended to increase the web page's interactivity, speed, functionality, and usability. Read more about Ajax from <http://www.w3schools.com/ajax/default.asp>

## 3.20 Miscellaneous

### 3.20.1 Collection Sorting

Collection class provides overloaded methods for sorting collections. Note that Collection Class is available only for .NET 1.1.

```
// Sorts the collection according to primary attributes.  
public new void Sort()
```

```
// Sorts the collection according to Attribute.  
public void Sort(AttributeComparer comparer)
```

```
// Sorts the collection according to Attribute in specified  
// range.  
public void Sort(int index, int count, AttributeComparer comparer)
```

The following code example illustrates how to sort CustomersCollection by attribute City.

[Visual Basic]

```
Dim csColl As CustomersCollection
Dim objFactory As ICustomersFactory =
    ServiceProvider.getCustomersFactory()
csColl = objFactory.ShowAll()

`Instantiate a new Attribute Comparer
AttributeComparer comparer = new AttributeComparer();

`Add attribute in comparer by which you want to sort the collection
comparer.AddAttribute("City")

`Sort the collection
csColl.Sort(comparer)
```

### 3.20.2 Custom Formula Field

TierDeveloper lets you define custom formula fields in an object that can span over one or more table attributes. For example you can define an "AllUnitsPrice" attribute in the "Product" Object that calculates the total price for all units of a specific product.

In that case, the formula field becomes part of the generated SQL as illustrated below and is thus calculated as part of the query execution.

```
SELECT t1.ProductID t1_ProductID, t1.ProductName t1_ProductName,
t1.UnitPrice t1_UnitPrice, t1.UnitsInStock t1_UnitsInStock,
t1.UnitsInStock * t1.UnitPrice Products_AllUnitsPrice
```

The generated component code provides the accessor method for a formula field as illustrated below. You can define as many formula fields as you need.

```
public Decimal AllUnitsPrice
{
    get
    {
        if (!getField("AllUnitsPrice").Null)
            return (Decimal)
                getField("AllUnitsPrice").getValue();
        else
            return new Decimal();
    }
    set
    {
        getField("AllUnitsPrice").setValue(value);
    }
}
```

### 3.20.3 Import/ Export Support

TierDeveloper facilitates the Import/Export operation from the XML file and DDL Scripts.

An XML file has information about tables and their attributes. You can import or

export objects from an XML file, if the objects that you are importing have corresponding tables in the database that you have in your data source.

DDL Scripts are SQL scripts that can be used to create tables in your data source. They may be for unmapped Objects. When these scripts are run on the SQL server, tables based on these scripts are generated. Mostly user-defined tables are exported through DDLs.

## **3.21 Component Deployment**

User can deploy Serviced or Non-Serviced components as explained below:

### **3.21.1 Serviced Components**

Once the build process is completed, the generated components are deployed under COM+ as serviced components using Regsvcs.exe utility that comes with Microsoft .Net platform SDK.

After the deployment of serviced components, gacutil.exe is run. Global Assembly Cache utility (gacutil.exe) is a command-line interface tool provided in .NET utility library to manage the assembly cache. Here -i switch is used with to install the provided assembly file, provided if user has not selected the Non-GAC Non-MTS option.

### **3.21.2 Non-Serviced Components**

The deployment process for Non-Serviced components is straight forward. As part of the build process the generated components are copied to the bin folder.

## **3.22 Remoting Application Deployment**

To deploy Remoting application when client and server are on different machines, do the following:

- Give the server name in place of 'localhost' in remoting URL of 'ClientApp\bin\xxx.remoting.proxy.dll.config'.

## **3.23 WebServices Application Deployment**

When client and server are on different machines, follow these steps to deploy Webservice application:

- In wsdl.bat file, give name of the Server in place of 'Localhost'.
- Run wsdl.bat. It will generate proxy.
- Run build.bat. It will build the proxy and a dll will be created as a result.
- Place the dll of Proxy in the bin of client application.
- Add references of proxy in client application and build the application.